**🧪 Experiment: Lexical Analyzer using LEX**

**🎯 Aim:**

To **generate a lexical analyzer** for a **subset of the C language** using the **LEX tool**.

**📌 Objectives:**

1. Understand the **Lexical Analysis Phase** of a compiler.
2. Learn to implement a **scanner** for a **subset of C** using **LEX**.

**📖 Theory Summary:**

**🔤 1. Token, Lexeme, and Pattern:**

| **Term** | **Description** |
| --- | --- |
| **Token** | A category/type of lexemes, e.g., identifier, keyword, operator. |
| **Lexeme** | Actual character string matching a token pattern, e.g., int, main. |
| **Pattern** | Regular expression describing a set of lexemes for a token. |

**🔁 2. Regular Expressions in Lexical Analysis:**

* Regular expressions are used to define the **patterns** for tokens in programming languages.
* LEX uses these patterns to scan and match input characters into meaningful **tokens**.
* Examples:
  + int → keyword
  + [a-zA-Z\_][a-zA-Z0-9\_]\* → identifier
  + [0-9]+ → number

**📜 Lex Program Format (Sections):**

%{

/\* Declarations (C code, header files) \*/

%}

%%

/\* Rules: pattern { action } \*/

%%

/\* User-defined functions (e.g., main) \*/

**📥 Input:**

A C program (subset) with basic constructs like:

* Keywords (int, float, if, while)
* Identifiers
* Numbers
* Operators
* Symbols ({, }, ;, etc.)

**📤 Output:**

1. **List of Tokens** with corresponding lexemes.
2. **Symbol Table** for identifiers (optional enhancement).

**✅ Conclusion:**

Successfully implemented a **scanner** for the **subset of C** using **LEX**, identifying tokens like keywords, identifiers, numbers, and operators.

**💻 Platform:**

Linux / Windows (via WSL or Cygwin with LEX/YACC installed)

**🎤 Viva Questions & One-Line Answers**

**🧠 Lex Tool Basics**

1. **What is LEX used for?**  
   → LEX is used to generate lexical analyzers (scanners) from regular expressions.
2. **What is a lexical analyzer?**  
   → It's the first phase of a compiler that scans source code and produces tokens.

**🧩 Conceptual Understanding**

1. **What is a token?**  
   → A token is a category of lexical units like identifier, number, or keyword.
2. **What is a lexeme?**  
   → A lexeme is the actual string from source code that matches a token pattern.
3. **What is a pattern in lexical analysis?**  
   → A regular expression that defines a class of lexemes for a token.

**🧾 Lex Specification Format**

1. **What is the structure of a Lex file?**  
   → It has three sections: definitions, rules, and user subroutines.
2. **What is the file extension of a Lex file?**  
   → .l (example: scanner.l)
3. **How do you compile and run a Lex program?**  
   → lex filename.l → cc lex.yy.c -o output → ./output

**⚙️ Lex Functions**

1. **What is yylex()?**  
   → It's the main function that matches input against defined token patterns.
2. **What is yywrap()?**  
   → It's called by yylex() on EOF; returns 1 to signal completion.
3. **What is yytext?**  
   → It’s a global char array holding the current lexeme matched.

**🧪 Examples and Extensions**

1. **How do you recognize an identifier in Lex?**  
   → Using [a-zA-Z\_][a-zA-Z0-9\_]\*
2. **How do you differentiate between keywords and identifiers?**  
   → Check for specific keywords first; otherwise, treat as identifiers.
3. **Can LEX be used for full parsing?**  
   → No, LEX only performs tokenizing; full parsing requires YACC or Bison.
4. **Can you create a symbol table in Lex?**  
   → Yes, by storing identifiers in a data structure during token recognition.

Main.l  
  
  
%{

#include <stdio.h>

#include <stdlib.h>

%}

%option noyywrap

%%

"int" { printf("KEYWORD: int\n"); }

"float" { printf("KEYWORD: float\n"); }

"char" { printf("KEYWORD: char\n"); }

"return" { printf("KEYWORD: return\n"); }

"if" { printf("KEYWORD: if\n"); }

"else" { printf("KEYWORD: else\n"); }

[0-9]+ { printf("NUMBER: %s\n", yytext); }

[a-zA-Z\_][a-zA-Z0-9\_]\* { printf("IDENTIFIER: %s\n", yytext); }

"+" { printf("OPERATOR: +\n"); }

"-" { printf("OPERATOR: -\n"); }

"\*" { printf("OPERATOR: \*\n"); }

"/" { printf("OPERATOR: /\n"); }

"=" { printf("ASSIGNMENT: =\n"); }

"(" { printf("PARENTHESIS: (\n"); }

")" { printf("PARENTHESIS: )\n"); }

"{" { printf("BRACE: {\n"); }

"}" { printf("BRACE: }\n"); }

";" { printf("SEMICOLON: ;\n"); }

[ \t\n] { /\* ignore whitespace \*/ }

. { printf("UNKNOWN CHARACTER: %s\n", yytext); }

%%

int main(int argc, char \*\*argv) {

yylex();

return 0;

}